TITLE:

複雑な個数制約の付いた多資源一般化割当問題について (最適化手法の深化と広がり)

AUTHOR(S):

小木曽, 由明; 今堀, 慎治; 柳浦, 睦憲

CITATION:

小木曽, 由明 ...[et al]. 複雑な個数制約の付いた多資源一般化割当問題について (最適化手法の深化と広がり). 数理解析研究所講究録 2012, 1773: 218-230

ISSUE DATE:

2012-01

URL:

http://hdl.handle.net/2433/171698

RIGHT:

京都大学学術情報リポジトリ
KURENAI 紅
Kyoto University Research Information Repository

数理解析研究所講究録
第1773巻 2012年 218-230

複雑な個数制約の付いた多資源一般化割当問題について

小木曽 由明[1] 今堀 慎治[2] 柳浦 睦憲[1]
[1]名古屋大学情報科学研究科計算機数理科学専攻
[2]名古屋大学工学研究科計算理工学専攻

概要 本研究では, 個数制約の付いた多資源一般化割当問題について考える. 個数制約付き多資源一般化割当問題とは, 仕事の割当個数と資源に関する制約条件の下で, 複数の仕事を複数のエージェントに割り当て, 割当に伴うコストを最小化する問題である. 個数制約は, 各エージェントに割り当てられた仕事数が指定された候補のいずれかでなければならないというものである. また, 資源制約は, 各エージェントに割り当てられた仕事が消費する資源量の合計がそのエージェントの資源容量を超えてはならないというものである. この問題は資源数が1以上の場合はNP困難であることが知られている. 資源数が0の場合については, 個数制約が特別な条件を満たす場合は多項式時間で解け, 一般の場合はNP困難であることを証明した. また, この問題に対して, 反復局所探索法に基づくアルゴリズムを提案した. 提案手法は, 大規模近傍探索法という大きな近傍を効率的に探索する手法を用いている. また, 実行不可能解を探索の対象に含め, そのような解の制約違反度を評価するためのペナルティの重みを適応的に更新することで, 実行可能領域と実行不可能領域をバランスよく探索する工夫を行っている. 一般化割当問題と多資源一般化割当問題のベンチマーク問題例を基に作成した問題例に対して, 提案アルゴリズムと汎用ソルバーCPLEXを適用し, 比較を行った. その結果, 問題例の多くにおいて提案アルゴリズムがCPLEXよりも良い結果を得ることを確認した.

1 はじめに

個数制約付き多資源一般化割当問題に対するメタ戦略アルゴリズムを提案する. 個数制約付き多資源一般化割当問題は, n 個の仕事を m 人のエージェントに割り当てる問題であり, 各エージェントが持つ複数の種類の資源に対する制約と割り当てられる仕事数に対する制約を満たしながら割当コストを最小化することが求められる. 個数制約付き多資源一般化割当問題は, 多資源一般化割当問題に個数制約を付加した問題である. また, 多資源一般化割当問題は資源の種類数が1であるとき一般化割当問題と呼ばれ, 古くから研究されている. 一般化割当問題はNP困難[11]であることが知られている. したがって, 個数制約付き多資源一般化割当問題もNP困難である.

一般化割当問題や多資源一般化割当問題に対するアルゴリズムは数多く存在する[6, 10, 13, 14, 15]. 一方, 本研究で取り扱う個数制約を付加した多資源一般化割当問題の研究は著者らが知る限りなされていない. しかし, この問題は, グループ分けなどに活用でき, 現実の応用が数多くあると考えられる.

本稿では, この問題に対して反復局所探索法に基づくアルゴリズムを提案する. 提案手法では, 大規模近傍探索法という大きな近傍を効率的に探索する手法を用いている. また, 実行不可能解を探索の対象に含め, そのような解の制約違反度を評価するためのペナルティの重みを適応的に更新することで, 実行可能領域と実行不可能領域をバランスよく探索する工夫を行っている. さらに, 解の探索を広く行うため, 反復局所探索法におけるキックの操作として, あるエージェントに割り当てられている仕事数を大きく変動させるシフトキックと, 各エージェントに割り当てられた仕事数を変えることなく複数の仕事の割当先を変動させる交換キックを組み込んでいる.

提案アルゴリズムの性能を評価するため, 一般化割当問題と多資源一般化割当問題のベンチマーク問題例を基に作成した問題例に対して, 提案アルゴリズムと汎用ソルバーCPLEXを適用し, 比較を行った. その結果, 問題例の多くにおいて提案アルゴリズムがCPLEXよりも良い結果を得ることを確認した.

本稿は以下のように構成されている. 2章で個数制約付き多資源一般化割当問題の説明と計算の複雑さの証明を行ったのち, 3章で提案アルゴリズムを説明する, 4章では計算結果を報告し, 5章でこれらについてのまとめを述べる.

2 個数制約付き多資源一般化割当問題

2.1 問題の定義

個数制約付き多資源一般化割当問題の定義を以下に与える．入力としてエージェントの集合 $I = \{1, \ldots, m\}$ とそれらに割り当てる仕事の集合 $J = \{1, \ldots, n\}$, および資源集合 $K = \{1, \ldots, s\}$ が与えられる．また, 各 $i \in I$, $j \in J$, $k \in K$ に対して仕事 j をエージェント i に割り当てたときにかかるコスト c_{ij}, 仕事 j をエージェント i に割り当てたときに使用する資源 k の量 a_{ijk}, エージェント i が持つ資源 k に対する資源容量 b_{ik} が与えられる．各資源 k に関する制約条件として, 各エージェント i に割り当てられた仕事が使用する資源 k の量 a_{ijk} の合計がそのエージェントの資源容量 b_{ik} を超えてはならないという資源制約がある．また, 仕事の割当個数に関する制約として, エージェント i に割り当てることが可能な仕事数の集合 H_i が与えられる．例えば, あるエージェント i に割当可能な仕事数が「2 個, 5 個, 7 個」のいずれかであるとき, $H_i = \{2, 5, 7\}$ となる．以降では便宜上, 集合 H_i の要素を小さい順に $h_{i1}, h_{i2}, \ldots$ と記す ($h_{i1} < h_{i2} < \cdots$)．つまり上の例の場合, $h_{i1} = 2$, $h_{i2} = 5$, $h_{i3} = 7$ となる．個数制約とは, 各エージェント i に割り当てられた仕事数がそのエージェントに割り当てることが可能な仕事数の集合 H_i のいずれかの要素と一致しなければならないという制約である．この問題の目的は, 資源制約と個数制約を満たしながら各仕事をちょうど一つのエージェントに割り当て, 割当に伴うコストの合計を最小化することである．出力は仕事のエージェントへの割り当て σ: $J \to I$ である．$\sigma(j) = i$ は, 仕事 j がエージェント i に割り当てられていることを表す．なお, この問題から個数制約を除いた問題を多資源一般化割当問題といい, さらに資源の種類数が 1 つであるとき一般化割当問題という．

2.2 定式化

$i \in I$ と $j \in J$ の組に対して, 仕事 j をエージェント i に割り当てる時に $x_{ij} = 1$, そうでない時に $x_{ij} = 0$ の値をとる 0-1 変数 x_{ij} を用意する．すなわち, $x_{ij} = 1 \Leftrightarrow \sigma(j) = i$ である．これを用いると, 個数制約付き多資源一般化割当問題は以下のように定式化出来る:

$$\text{minimize} \quad cost(\sigma) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ijk} x_{ij} \leq b_{ik}, \quad \forall i \in I, \ \forall k \in K, \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} \in H_i, \quad \forall i \in I, \tag{3}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \forall j \in J, \tag{4}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in I, \ \forall j \in J.$$

目的関数 (1) は各仕事をいずれかのエージェントに割り当てた時のコストの総和を表している．また, 制約式 (2) は資源制約を表しており, 左辺はエージェント i に割り当てられた仕事が使用する資源 k の合計量, 右辺はエージェント i の資源 k に対する容量である．制約式 (3) は個数制約を表しており, 左辺はエージェント i に割り当てられた仕事数, 右辺は割り当て可能な仕事数の集合である．制約式 (4) は各仕事が丁度一人のエージェントに割り当てられることを表している．

2.3 問題の計算の複雑さ

資源数 $s \geq 1$ の場合には多資源一般化割当問題は NP 困難であることが知られているため, 本問題は NP 困難である．

一方, 資源数 $s = 0$ の場合, たとえば割当可能な仕事数が n 以下の偶数 (あるいは奇数) 全ての集合であるような特殊な個数制約を持つ場合にはこの問題は多項式時間で解ける．より一般的に, 以下が成り立つ．

定理 1. 資源数 $s = 0$ の場合, 各エージェント $i \in I$ の割当可能な仕事数が $[2l_i, 2r_i]$ 内の全ての偶数の集合か, $[2l_i + 1, 2r_i + 1]$ 内の全ての奇数の集合, または $[l_i, r_i]$ 内の全ての値の集合のいずれかとなるとき, 個数制約付き多資源一般化割当問題は多項式時間で解くことが出来る．ただし l_i と r_i はそれぞれ $0 \leq l_i \leq r_i$ を満たす任意の整数である．

証明. この問題を最小コスト完全マッチング問題に帰着することで証明する．

まず, 最小コスト完全マッチング問題について以

下に述べる．入力として，n 個の頂点と m 個の辺を持つ無向グラフ $G=(V,E)$ が与えられる．また，各辺 (u,v) は重み $w(u,v)$ を持つ．出力は辺の部分集合 $M \subseteq E$ である．ここで，部分集合 M に対して，M 内のどの 2 つの異なる辺も端点を共有しないとき，M はマッチングであるという．また，頂点 $v \in V$ がある $e \in M$ の端点であるとき，v はマッチされているといい，とくに全頂点がマッチされているマッチングを完全マッチングという．完全マッチングに含まれる辺 $(u,v) \in M$ の重み和をそのマッチングに対するコストという．この問題の目的は，コストが最小となる完全マッチングを求めることである．最小コスト完全マッチング問題は多項式時間で解けることが古くから知られており [3]，Lawler の $O(n^3)$ 時間アルゴリズム [7]，Gabow の $O(mn+n^2\log n)$ 時間アルゴリズム [4]，Mehlhorn と Schäfer の $O(nm\log n)$ 時間のアルゴリズム [9]，Cook と Rohe の $O(n^3)$ 時間のアルゴリズム [2] などがある．

便宜上，個数制約が $[2l_i, 2r_i]$ 内の全ての偶数の集合となるエージェントの集合を I_{even}，$[2l_i+1, 2r_i+1]$ 内の全ての奇数の集合となるエージェントの集合を I_{odd}，$[l_i, r_i]$ 内の全ての値の集合となるエージェントの集合を I_{int} とする ($l_i = r_i \Rightarrow i \in I_{\text{int}}$ とする)．つまりエージェント $i \in I_{\text{even}}$ の個数制約は $H_i = \{2l_i, 2l_i+2, \ldots, 2r_i\}$，$i \in I_{\text{odd}}$ の個数制約は $H_i = \{2l_i+1, 2l_i+3, \ldots, 2r_i+1\}$，$i \in I_{\text{int}}$ の個数制約は $H_i = \{l_i, l_i+1, \ldots, r_i\}$ である．このような問題例に対して最小コスト完全マッチング問題の問題例を以下のように構成する．まず，各仕事に対応した頂点集合 $V_1 = \{v_1, v_2, \ldots, v_n\}$ を用意する．次に，各エージェントに対応する頂点集合 $V_2 = V_{\text{even}} \cup V_{\text{odd}} \cup V_{\text{int}}$ を以下のように用意する:

$$
\begin{aligned}
V_{\text{even}} &= \{u_1^i, u_2^i, \ldots, u_{2r_i}^i \mid i \in I_{\text{even}}\} \\
V_{\text{odd}} &= \{u_1^i, u_2^i, \ldots, u_{2r_i+1}^i \mid i \in I_{\text{odd}}\} \\
V_{\text{int}} &= \{u_1^i, u_2^i, \ldots, u_{r_i}^i \mid i \in I_{\text{int}}\}.
\end{aligned}
$$

また，ダミーの頂点 $V_{\text{dummy}} = \{z_1, z_2, \ldots, z_\tau\}$ を設ける．これらの頂点は仕事に対応する頂点 $v \in V_1$ とマッチしなかった頂点 $u \in V_{\text{int}}$ をマッチさせるために用意する．$\kappa(\nu)$ を ν 以上の最小の偶数と定義すると，用意するダミー頂点の数 τ は次のようになる:

$$
\nu = |V_{\text{int}}| - \sum_{i \in I_{\text{int}}} l_i,
$$

$$
\chi = n - \left\{ \sum_{i \in I_{\text{int}}} l_i + \sum_{i \in I_{\text{even}}} 2l_i + \sum_{i \in I_{\text{odd}}} (2l_i+1) \right\},
$$

$$
\tau = \begin{cases} \kappa(\nu), & \nu - \chi\text{が偶数である}, \\ \kappa(\nu)-1, & \text{その他}. \end{cases}
$$

次に，頂点集合 $V = V_1 \cup V_2 \cup V_{\text{dummy}}$ に対して以下のように辺集合 $E = E_1 \cup E_{\text{even}} \cup E_{\text{odd}} \cup E_{\text{int}} \cup E_{\text{dummy}}$ を定義する:

$$
\begin{aligned}
E_1 &= \{(v_j, u_d^i) \mid v_j \in V_1, u_d^i \in V_2\}, \\
E_{\text{even}} &= \{(u_d^i, u_{d+1}^i) \mid i \in I_{\text{even}}, \\
&\qquad d = 2l_i+1, 2l_i+2, \ldots, 2r_i-1\}, \\
E_{\text{odd}} &= \{(u_d^i, u_{d+1}^i) \mid i \in I_{\text{odd}}, \\
&\qquad d = 2l_i+2, 2l_i+3, \ldots, 2r_i\}, \\
E_{\text{int}} &= \{(u_d^i, z_t) \mid i \in I_{\text{int}}, \\
&\qquad d = l_i+1, l_i+2, \ldots, r_i,\ t = 1, 2, \ldots, \tau\}, \\
E_{\text{dummy}} &= \{(z_t, z_{t+1}) \mid t = 1, 2, \ldots, \tau-1\}.
\end{aligned}
$$

ここで，E_1 は仕事に対応する頂点とエージェントに対応する頂点間の辺で，E_{even} (E_{odd}) は集合 I_{even} (I_{odd}) 内の同一のエージェントに対応する隣り合う頂点間の辺である．また，E_{int} は集合 I_{int} 内のエージェントに対応する頂点とダミーの頂点間の辺で，E_{dummy} は隣り合うダミーの頂点間の辺である．また，各辺の重みを以下のように定める．

$$
\begin{aligned}
w(v_j, u_d^i) &= c_{ij}, & &\forall (v_j, u_d^i) \in E_1, \\
w(u_d^i, u_{d+1}^i) &= 0, & &\forall (u_d^i, u_{d+1}^i) \in E_{\text{even}} \cup E_{\text{odd}}, \\
w(u_d^i, z_t) &= 0, & &\forall (u_d^i, z_t) \in E_{\text{int}}, \\
w(z_t, z_{t+1}) &= 0, & &\forall (z_t, z_{t+1}) \in E_{\text{dummy}}.
\end{aligned}
$$

以下では，任意の完全マッチング M から，コストが等しく実行可能な仕事の割当 σ が定義できることを示す．まず，頂点集合 V_1 と V_2 の間の辺 $(v_j, u_d^i) \in M$ ($v_j \in V_1$, $u_d^i \in V_2$) に対応して，エージェント i に仕事 j を $\sigma(j) = i$ と割り当てる．ここで，完全マッチングの定義から各頂点 $v_j \in V_1$ は丁度ひとつの頂点 $u_d^i \in V_2$ とマッチするため，どの仕事も丁度一人のエージェントに割り当てられる．さらに，この割当にかかるコスト c_{ij} の総和は，辺 $(v_j, u_d^i) \in M$ に対応する重み $w(v_j, u_d^i)$ の総和となるため，マッチング M の辺重みの総和と一致する．また，任意の頂点集合 $\{u_1^i, u_2^i, \ldots, u_{2r_i}^i\} \subseteq V_{\text{even}}$ に対して，辺集合 E の定義から頂点 $u_1^i, u_2^i, \ldots, u_{2l_i}^i$ はそれぞれ頂点 $v \in V_1$ との辺しか持たない．そのため M においてこれらの頂点は必ず V_1 の頂点

とマッチされている．さらに，頂点の定義からエージェント i に対応する頂点の数はエージェント i に割当可能な仕事数の上界 $2r_i$ までしか用意されていない．以上のことから，エージェント i に割り当てられた仕事数は $2l_i$ 以上 $2r_i$ 以下となる．また，頂点集合 $\{u_1^i, u_2^i, \ldots, u_{2r_i}^i\} \subseteq V_{\text{even}}$ のうち，頂点 $u_{2l_i+1}^i, u_{2l_i+2}^i, \ldots, u_{2r_i}^i$ は偶数個である．M においてはこれらのうち偶数個の頂点が V_1 とマッチされ，残りの偶数個の頂点同士がマッチされているため，対応するエージェント i には偶数個の仕事が割り当てられる．よって $\{u_1^i, u_2^i, \ldots, u_{2r_i}^i\} \subseteq V_{\text{even}}$ に対応するエージェント $i \in I_{\text{even}}$ には $2l_i$ 以上 $2r_i$ 以下の偶数個の仕事が割り当てられる．V_{odd} に対しても同様となる．また，任意の頂点集合 $\{u_1^i, u_2^i, \ldots, u_{r_i}^i\} \subseteq V_{\text{int}}$ に対して，頂点 $u_1^i, u_2^i, \ldots, u_{l_i}^i$ は頂点集合 V_1 のいずれかの頂点とマッチし，残りの頂点は $V_1 \cup V_{\text{dummy}}$ のいずれかの頂点とマッチするため，対応するエージェント i に割り当てられる仕事数は l_i 以上 r_i 以下となる．これらのことより，割当 σ は個数制約を満たしている．すなわち実行可能な仕事の割当を得られることが確認できた．

次に，実行可能な任意の仕事の割当 σ からコストの等しい完全マッチング M が定義できることを示す．$M = \emptyset$ から始め，$J = 1, 2, \ldots, n$ の順に以下の手順に従って M に辺を追加していく．仕事 j のエージェント $i = \sigma(j)$ への割当に対応して，仕事 j に対応する頂点 $v_j \in V_1$ と，エージェント i に対応する頂点 $u_1^i, u_2^i, \ldots$ のうちマッチしていないものの中で最小の添え字 d を持つ u_d^i をマッチさせ，$M := M \cup \{(v_j, u_d^i)\}$ とする．どの仕事もいずれか一人のエージェントに割り当てられているため，頂点集合 V_1 に含まれる頂点は頂点集合 V_2 のいずれかの頂点とマッチすることとなる．また，割当 σ が個数制約を満たしていることから，V_2 の中で $E_{\text{even}} \cup E_{\text{odd}} \cup E_{\text{int}}$ の辺に接続していない頂点 ($\{u_1^i, u_2^i, \ldots, u_{2l_i}^i\} \subseteq V_{\text{even}}$ など) は全て V_1 の頂点とマッチしている．次に，頂点集合 $\{u_{2l_i+1}^i, \ldots, u_{2r_i}^i\} \subseteq V_{\text{even}}$ の中で頂点集合 V_1 とマッチしていない頂点について考える．そのような頂点の数は偶数であり，添え字は連続している．よってこれらの頂点は全て E_{even} の辺を用いてマッチさせることが出来る．これらの辺を M に追加する．V_{odd} についても同様である．次に，頂点集合 V_{int} の中で頂点集合 V_1 とマッチしていない頂点を V_{dummy} の頂点に添え字の小さい順にマッチしていき，対応する辺を M に追加する．その結果，頂点集合 V_{dummy} の中で V_{int} にマッチされずに残る頂点の数が偶数であることを容易に示すことが出来る．そのため，これら残りの頂点同士を E_{dummy} の辺を用いてマッチさせることで，V_{dummy} の全ての頂点をマッチすることが出来る．これらの辺を M に追加する．以上より任意の実行可能な仕事の割当 σ から完全マッチング M を定義できる．このように定義したマッチング M のコストが割当 σ のコストと同じであることは自明である．

以上より，個数制約付き割当問題と最小コスト完全マッチング問題は最適値が一致すること，および最小コスト完全マッチング問題の任意の実行可能解から同じコストを持つ実行可能な割当を作成できることが証明出来た．問題例および解の変換に必要な計算量が多項式時間であることは自明である．したがって，定理の条件を満たす個数制約付き多資源一般化割当問題を最小コスト完全マッチング問題に帰着出来た． □

資源数 $s = 0$ の場合でも，一般にはこの問題はNP 困難である．各エージェントに割当可能な仕事数が 2 種類に限定された単純な個数制約であってもNP 困難であることがいえる．

定理 2. 資源数 $s = 0$ の場合，各エージェント $i \in I$ に割当可能な仕事数が 0 と 3 に限定されている場合でも，個数制約付き多資源一般化割当問題は NP 困難である．

証明. 集合分割問題をこの問題に帰着することで証明する．

集合分割問題について以下に述べる．入力として集合 $D = \{1, 2, \ldots, n\}$ の部分集合 S_i $(i = 1, 2, \ldots, m)$ が与えられる．これらの中から S_i の要素が重複しないようにいくつかの S_i を選ぶ．そのような S_i の組合せの中で，和集合が D となるようなものが存在するかどうかを問う問題である．すなわち，部分集合の添字集合 $X \subseteq \{1, 2, \ldots, m\}$ で

$$S_i \cap S_{i'} = \emptyset, \ \forall i \neq i' \in X$$
$$\bigcup_{i \in X} S_i = D$$

を満たすものがあるか否かを判定する問題で，この問題はNP 困難であることが知られている．また，集合分割問題は各部分集合 S_i の要素数が 3 つに限定され

ているとき, exact cover by 3-sets と呼ばれるが, この問題も NP 困難であることが知られている [5].

まず, 集合 D に対応して n 個の仕事を用意する. 次に, 部分集合 $S_1, S_2, \ldots, S_m$ に対応してエージェント $1, 2, \ldots, m$ を用意する. 割当コスト c_{ij} は $j \in S_i$ ならば 0, そうでないならば 1 とする. また, 各エージェント i の個数制約を $H_i = \{0, |S_i|\}$ とする.

まず, 集合分割問題の制約を満たす解 X が存在する場合, コスト 0 の仕事の割当が存在することを示す. 解 X に含まれる各部分集合 S_i に対応するエージェント i に部分集合の要素 $j \in S_i$ に対応する仕事を全て割り当てる. このとき定義より割当コストは 0 となる. さらに集合分割問題の定義より各仕事は丁度一人のエージェントに割り当てられ, 各エージェント i に割り当てられた仕事数は 0 か $|S_i|$ となり個数制約も満たしている. よって, コスト 0 の仕事の割当が存在する.

次に, コスト 0 の仕事の割当 σ が存在する場合, 集合分割問題の制約を満たす解が存在することを示す. 各エージェント i に対して, σ において i に割り当てられた仕事数が正であるとき, かつそのときに限り部分集合 S_i を集合分割問題の解に含める. すなわち, $X = \{i \mid |\{j \mid \sigma(j) = i\}| \geq 1\}$ とする. 割当 σ のコストが 0 であることから, 任意の仕事 j とその割当先 $i = \sigma(j)$ に対して $j \in S_i$ が成り立つ. また, σ が個数制約を満たしていることから, 各エージェント i に割り当てられた仕事数は 0 か $|S_i|$ であり, 割り当てられた仕事数が $|S_i|$ であるエージェント i には部分集合 S_i の要素に対応する仕事が全て割り当てられている. σ においてはどの仕事も丁度一人のエージェントに割り当てられていることから, 集合 D の各要素は X の中の丁度ひとつの部分集合に含まれている. よって, X は集合分割問題の制約を満たす. 以上のことから, 集合分割問題を個数制約付き多資源一般化割当問題に帰着出来た. □

3 提案アルゴリズム

3.1 提案アルゴリズムの概要

提案アルゴリズムは組合せ最適化問題に対する基本的な戦略の 1 つである局所探索法に基づいており, 反復局所探索法を基本としている.

局所探索法は, 適当な初期解 σ から始め, 近傍 $N(\sigma)$ 内に改善解 σ' があれば $\sigma := \sigma'$ とする操作を, 近傍内に改善解が見つからなくなるまで繰り返す方法である. ここで, 近傍とは解にある操作を加えることにより得られる解集合のことをいい, 近傍の定義の仕方に探索の効率は大きく影響する. 局所探索の結果得られた解を局所最適解といい, 局所最適解の近傍内には改善解が存在しないことを意味している. 一般化割当問題に対する局所探索の近傍として, シフト近傍と交換近傍が用いられることが多い. ここで, シフト近傍 $N_{\text{shift}}(\sigma)$ と交換近傍 $N_{\text{swap}}(\sigma)$ は

$$N_{\text{shift}}(\sigma) = \{\sigma' \mid \text{解}\sigma'\text{は解}\sigma\text{のひとつの仕事の割当先を変更することにより得られる}\},$$
$$N_{\text{swap}}(\sigma) = \{\sigma' \mid \text{解}\sigma'\text{は解}\sigma\text{のふたつの仕事の割当先を交換することにより得られる}\}$$

である. これらの近傍サイズはシフト近傍が $O(mn)$, 交換近傍が $O(n^2)$ である. 提案アルゴリズムでは, これらの通常の近傍に加えて連鎖シフト近傍を用いた. 連鎖シフト近傍 $N_{\text{chain}}(\sigma)$ とは l $(l = 2, 3, \ldots, n)$ 個の仕事 $j_1, j_2, \ldots, j_l$ の割当先を

$$\sigma'(j_r) := \sigma(j_{r-1}),\ r = 2, 3, \ldots, l,$$
$$\sigma'(j_1) := \sigma(j_l)$$

と変更することによって得られる解 σ' の集合である. つまり, $r = 2, 3, \ldots, l$ に対して, 仕事 j_r の割当先をエージェント $\sigma(j_r)$ から $\sigma(j_{r-1})$ に変更する. そして, 仕事 j_1 の割当先をエージェント $\sigma(j_l)$ に変更するということである. この連鎖シフト近傍の近傍サイズは $O(n^l)$ であり, l の指数関数的に増加していく. そのため, 改善グラフと呼ばれるアイデアを用いて探索対象となる解を制限する. これら 3 つの近傍サイズは $|N_{\text{shift}}| \leq |N_{\text{swap}}| \leq |N_{\text{chain}}|$ となるため, N_{shift} の近傍内に改善解が見つからないときのみ N_{swap} の探索を行い, また N_{shift} と N_{swap} のいずれの近傍内にも改善解が見つからないときのみ N_{chain} の探索を行う.

提案手法では, 探索空間を任意の割当 σ の集合とする. すなわち制約を満たさない実行不可能解も探索の対象に含めて探索を行う. ここで, 割当 σ において, 任意のエージェント i に割り当てられた仕事の集合を

$$J_i^\sigma = \{j \in J \mid \sigma(j) = i\}$$

と定義する．局所探索の中で実行不可能領域を探索する際，以下のように実行可能解からの各制約の違反度に応じてペナルティを目的関数に付加した $pcost(\sigma)$ の値で解を評価することにする:

$$pcost(\sigma) = cost(\sigma) + \sum_{i=1}^{m}\sum_{k=1}^{s}(\alpha_{ik}p_{ik}(J_i^\sigma) + \beta_i q_i(J_i^\sigma)),$$

$$p_{ik}(S) = \max\left\{0, \sum_{j\in S} a_{ijk} - b_{ik}\right\}, \quad \forall i \in I,\ \forall k \in K,\ \forall S \subseteq J$$

$$q_i(S) = \min\{||S| - h_{ig}| \mid h_{ig} \in H_i\}, \quad \forall i \in I,\ \forall S \subseteq J.$$

ここで，パラメータ $\alpha_{ik}(>0)$ と $\beta_i(>0)$ は各制約に対するペナルティの重みであり，探索の間に適応的に更新していく．

反復局所探索法とは，過去の局所探索で得られた局所最適解の中の良い解 σ_{seed} にランダムな摂動を加えたものを初期解として，局所探索を反復する方法である．提案アルゴリズムでは，局所最適解 σ_{lopt} に辿り着いたとき，以下のルールに従って新たな初期解を生成する．まず，$pcost(\sigma_{\text{lopt}}) \leq pcost(\sigma_{\text{seed}})$ (ただし，最新のパラメータ α_{ik}, β_i に対して $pcost$ を評価する) であるとき，ランダムな摂動を加える解 σ_{seed} を $\sigma_{\text{seed}} := \sigma_{\text{lopt}}$ と更新する．提案アルゴリズムでは，ランダムな摂動としてシフトキックと交換キックを用いた．シフトキックは任意のエージェント i に割り当てられている複数の仕事の割当先を変更する操作であり，交換キックは異なるエージェントに割り当てられたふたつの仕事の割当先の交換を複数回行う操作である．これらシフトキックと交換キックの一方をランダムに選び，解 σ_{seed} に適用する．

以下の節で提案アルゴリズムの詳細について述べる．3.2 節では改善グラフについて述べ，3.3 節では改善グラフを用いた連鎖シフト近傍の探索について，3.4 節ではペナルティ重みの更新方法，3.5 節ではシフトキックと交換キックについて述べる．最後に，3.6 節では提案アルゴリズム全体の枠組みをまとめる．

3.2 改善グラフ

改善グラフ $G(\sigma) = (V, E)$ は，頂点集合 V と辺集合 E を持つ有向グラフである．各頂点 $j \in V$ が仕事 $j \in J$ に対応しており，$V = J$ となる．各辺 $(j_1, j_2) \in E$ はエージェント $\sigma(j_1)$ から仕事 j_1 を外し，仕事 j_2 をエージェント $\sigma(j_1)$ に割り当てることに対応しており，

$$E = \{(j_1, j_2) \mid j_1, j_2 \in V, \sigma(j_1) \neq \sigma(j_2)\} \quad (5)$$

となる．また，各辺 (j_1, j_2) は重み $w_{j_1j_2}$ を持ち，これはエージェント $\sigma(j_1)$ から仕事 j_1 を外し，仕事 j_2 をエージェント $\sigma(j_1)$ に割り当てたときのエージェント $\sigma(j_1)$ のコストとペナルティの変化量に対応する．仕事 $j_1, j_2, \ldots, j_l$ に対する連鎖シフト操作は，改善グラフ $G(\sigma)$ 内のサイクル $j_1 \to j_2 \to \cdots \to j_l \to j_1$ に対応する．ここでサイクルの辺の数は，対応する連鎖シフト操作において割当先を移動する仕事の数となる．簡便のために，

$$i_r = \sigma(j_r),\ r = 1, 2, \ldots, l,$$

とし，$j_{l+1} = j_1$, $i_{l+1} = i_1$ とする．また，任意のエージェント $i \in I$ と仕事の任意の部分集合 $S \subseteq J$ に対して

$$pcost_i(S) = \sum_{j\in S} c_{ij} + \sum_{k=1}^{s}(\alpha_{ik}p_{ik}(S) + \beta_i q_i(S))$$

と定義する．辺 $(j_1, j_2) \in E$ の重み $w_{j_1j_2}$ は

$$w_{j_1j_2} = pcost_{i_1}(J_{i_1}^\sigma \cup \{j_2\} \setminus \{j_1\}) - pcost_{i_1}(J_{i_1}^\sigma),$$

となる．また，連鎖シフト操作は任意のエージェントに対してひとつの仕事を除き，新しい仕事を割り当てることの繰り返しとなるため，各エージェントに割り当てられた仕事数に変化はない．つまり，$|J_{i_1}^\sigma \cup \{j_2\} \setminus \{j_1\}| = |J_{i_1}^\sigma|$ である．ここで，

$$\tilde{J}_{rr'}^\sigma = J_{i_r}^\sigma \cup \{j_{r'}\} \setminus \{j_r\}$$

と定義すると，辺の重み $w_{j_1j_2}$ の計算は，

$$\begin{aligned} w_{j_1j_2} &= pcost_{i_1}(\tilde{J}_{12}^\sigma) - pcost_{i_1}(J_{i_1}^\sigma) \\ &= \sum_{j\in\tilde{J}_{12}^\sigma} c_{i_1j} + \sum_{k=1}^{s}(\alpha_{i_1k}p_{i_1k}(\tilde{J}_{12}^\sigma) + \beta_{i_1}q_{i_1}(\tilde{J}_{12}^\sigma)) \\ &\quad - \sum_{j\in J_{i_1}^\sigma} c_{i_1j} - \sum_{k=1}^{s}(\alpha_{i_1k}p_{i_1k}(J_{i_1}^\sigma) + \beta_{i_1}q_{i_1}(J_{i_1}^\sigma)) \\ &= \sum_{j\in\tilde{J}_{12}^\sigma} c_{i_1j} - \sum_{j\in J_{i_1}^\sigma} c_{i_1j} \\ &\quad + \sum_{k=1}^{s}(\alpha_{i_1k}p_{i_1k}(\tilde{J}_{12}^\sigma) - \alpha_{i_1k}p_{i_1k}(J_{i_1}^\sigma)) \end{aligned}$$

$$= c_{i_1 j_2} - c_{i_1 j_1} + \sum_{k=1}^{s} (\alpha_{i_1 k} p_{i_1 k}(\tilde{J}^{\sigma}_{12}) - \alpha_{i_1 k} p_{i_1 k}(J^{\sigma}_{i_1})) \quad (6)$$

となり, 実際はコストと資源制約に対するペナルティの変化量の計算のみで良い. 各 i と k の組みに対して現在の解における資源使用量の合計を記憶しておくことで, 式 (6) から 1 つの枝あたり $O(s)$ 時間で重み $w_{j_1 j_2}$ を計算することが出来る.

ここで, 仕事 $j_1, j_2, \ldots, j_l$ に対する連鎖シフト操作と改善グラフ $G(\sigma)$ 内の対応するサイクル $j_1 \to j_2 \to \cdots \to j_l \to j_1$ について考える. 任意の r, r' $(r \neq r' \in \{1, 2, \ldots, l\})$ に対して $i_r \neq i_{r'}$ ならば, そのサイクルを subset-disjoint と呼ぶ. サイクルが subset-disjoint であるとき, 連鎖シフト操作による $pcost$ の変化量は

$$\sum_{r=1}^{l} pcost_{i_r}(\tilde{J}^{\sigma}_{r,r+1}) - \sum_{r=1}^{l} pcost_{i_r}(J^{\sigma}_{i_r}) = \sum_{r=1}^{l} \{pcost_{i_r}(\tilde{J}^{\sigma}_{r,r+1}) - pcost_{i_r}(J^{\sigma}_{i_r})\} = \sum_{r=1}^{l} w_{j_r j_{r+1}}$$

となる. このように, 連鎖シフト操作による $pcost$ の変化量は, 対応するサイクルが subset-disjoint であるとき, サイクルの辺の重み和となる. よって, 連鎖シフト近傍内の改善解を発見するには重み和が負となる subset-disjoint のサイクルを発見できれば良い. しかしながら, このようなサイクルを発見する問題は一般的に NP 困難である [12]. 一方, サイクルが subset-disjoint であるかどうかを考慮せず, 単に重み和が負となるサイクルを発見するのは多項式時間で行うことが出来る [1]. そのため, 改善グラフ内から重み和が負となるサイクルを発見することで, 連鎖シフト近傍内の改善解を近似的に探索していく.

改善グラフの探索効率は辺の数に依存するため, 以下のルールで辺の数を制限する. エージェント $i_1 = \sigma(j_1)$ から仕事 j_1 を外し, 仕事 j_2 を加えたとき, エージェント i_1 の資源容量を超える資源の種類数を

$$\mu_{j_1 j_2} = |\{k \in K \mid p_{i_1 k}(J^{\sigma}_{i_1} \setminus \{j_1\} \cup \{j_2\}) > 0\}|,$$

とする. そして辺を以下のように制限したグラフ $\tilde{G}(\sigma) = (V, \tilde{E})$ を考える.

$$\tilde{E} = \{(j_1, j_2) \in E \mid \mu_{j_1 j_2} = \min_{j' \in J} \mu_{j_1 j'}\}.$$

これは, 頂点 j_1 から出る各辺の中で資源容量を超える資源数が最も少ない辺のみに制限するということである. また, $(j_1, j_2) \notin \tilde{E}$ となる場合, $w_{j_1 j_2} = +\infty$ とする. この制限により, 近傍内の改善解を逃す可能性は大きくなってしまうが, その分探索の効率が向上することが期待出来る.

3.3 改善グラフの探索

本節では, 改善グラフ $\tilde{G}(\sigma) = (V, \tilde{E})$ を用いた連鎖シフト近傍の探索について述べる. ここで述べる手法は, サイクルの探索を subset-disjoint であるものに限定しない. したがって, サイクルの辺の重み和とサイクルに対応する連鎖シフト操作によるペナルティ付きコスト $pcost$ の変化量が必ずしも一致するとは限らない. ただし, 辺の数が 3 以下のサイクルは式 (5) の定義より, 常に subset-disjoint となる. また, 辺数の短いサイクルは subset-disjoint になりやすい傾向にあり, サイクル内の辺数が多くなるほど subset-disjoint である確率は減少していく. とくに辺数 $m+1$ 以上のサイクルでは, subset-disjoint となることはない. そのため, 辺数 m 以下のサイクルに制限して探索を行う.

動的計画法に基づいてサイクルの探索を行う. ここで任意の $j_1, j \in J$, $l = 1, 2, \ldots, m-1$ に対して, $f^*(j_1, j, l)$ を頂点 j_1 から頂点 j までの辺数 l の最小重みパスとする. 任意の $j_1, j \in J$ に対して, $f^*(j_1, j, l)$ は

$$f^*(j_1, j, l) = \begin{cases} w_{j_1 j}, & l = 1, \\ \min_{j' \in J} \{f^*(j_1, j', l-1) + w_{j' j}\}, & l = 2, 3, \ldots, m-1 \end{cases} \quad (7)$$

と計算出来る. ここで, 式 (7) において最小の値を実現する j' を保持しておき, l から 1 までこの値を辿ることで, $f^*(j_1, j, l)$ を実現するパスを復元出来る.

$f^*(j_1, j, l) + w_{j j_1}$ は辺 (j, j_1) を含む辺数 $l+1$ のサイクルの中で最小の重み和を表す. したがって, $f^*(j_1, j, l) + w_{j j_1} < 0$ となる (j_1, j, l) を探索することで改善グラフ内の負の重みをもつサイクルを発見することが出来る. ただし, $f^*(j_1, j, l) + w_{j j_1} < 0$ と

なるサイクルを発見しても, そのサイクルが subset-disjoint でない場合は $pcost$ の変化量がサイクルの重み和と同じでないかもしれない. そのため, サイクルに対応する連鎖シフト操作による $pcost$ の変化量を再計算する必要がある. これらの動作を以下にまとめる. これを連鎖シフト近傍探索 (SCS) と呼ぶ.

連鎖シフト近傍探索 (SCS)

入力: 現在の解 σ.

出力: $N_{\mathrm{chain}}(\sigma)$ の中に改善解を発見できなければ "no," 発見できれば改善解 $\sigma' \in N_{\mathrm{chain}}(\sigma)$.

Step1 $S := J$ とする.

Step2 $S = \emptyset$ ならば "no"と出力し終了する. そうでなければランダムに仕事 $j_1 \in S$ を選び, $S := S \setminus \{j_1\}$, $l := 1$ とする.

Step3 全ての $j \in J$ に対して式 (7) を用いて $f^*(j_1, j, l)$ を計算する.

Step4 $f^*(j_1, j, l) + w_{jj_1}$ の増加順に $j \in J$ をソートする. この順番で各 $j \in J$ に対して, サイクルに対応する連鎖シフト操作を行ったときの $pcost$ の変化量を計算する. もし, 改善解 σ を発見したならば σ を出力し, 終了する.

Step5 $l \leq m-2$ ならば $l := l+1$ とし, Step3 へ. そうでなければ Step2 へ.

このアルゴリズムの計算時間を評価する. $f^*(j_1, j, l)$ の計算にかかる時間はアルゴリズム全体で $O(nm|\tilde{E}|)$ 時間である. また, Step4 のソートには 1 回あたり $O(n \log n)$ 時間かかり, 全体では $O(n^2 m \log n)$ 時間かかる. 長さ l のサイクルに対する $pcost$ の再計算には 1 つあたり $O(ls)$ 時間かかり, 全体では $O(n^2 m^2 s)$ 時間かかる. よって全体の計算時間は $O(nm(|\tilde{E}| + n \log n + nms))$ 時間となる.

次にこの探索の高速化について述べる. まず, 関数

$$\lambda(x) = \begin{cases} +\infty, & x \geq 0, \\ x, & x < 0 \end{cases}$$

を用いて, $f^*_-(j_1, j', l) = \lambda(f^*(j_1, j', l))$ と定義する. これは

$$f^*_-(j_1, j, l) = \begin{cases} \lambda(w_{j_1 j}), & l = 1, \\ \min\limits_{j' \in J} \{\lambda(f^*_-(j_1, j', l-1) + w_{j'j})\}, & \\ & l = 2, 3, \ldots, m-1 \end{cases} \tag{8}$$

によって計算出来るが, どの $l' = 2, 3, \ldots, l$ に対しても $f^*(j_1, j, l'-1)$ が負となるパス $j_1 \to j_2 \to \cdots \to j_l$ のみを探索をすることを意味している. つまり頂点 j_1 から頂点 j への各長さ l' の最小重みパス $f^*(j_1, j, l)$ を計算する際, $f^*(j_1, j', l-1) < 0$ となる頂点 j' から出る辺のみに制限して計算を行うということである. そのため, (8) 式は (7) 式よりも効率的に計算出来る. この制限を行っても, グラフ内に負のサイクルが存在するとき, そのようなサイクルの 1 つを発見することが出来る [8]. また, Step4 で $f^*(j_1, j, l) + w_{jj_1}$ の増加順にソートし, ソートした順番に $pcost$ の再計算を行う際も, $f^*(j_1, j, l) + w_{jj_1} < 0$ となる頂点 j のみを再計算の対象とする.

3.4 ペナルティ重みの更新

本節では, ペナルティ重み α_{ik} と β_i の更新方法について述べる. 提案アルゴリズムの動作は α_{ik}, β_i の値に大きく影響される. そのため, これらの値をより効果的な値に適宜更新していく必要がある.

初期値として, 各 $i \in I$, $k \in K$ に対して $\alpha_{ik} := \alpha_{\mathrm{init}}$, $\beta_i := \beta_{\mathrm{init}}$ とする. また, 局所最適解 σ_{lopt} にたどり着くたびにペナルティ重み α_{ik} と β_i の更新を行う. ここで, 任意の $i \in I$, $k \in K$ に対し, ペナルティ重みの更新に用いる関数を以下のように定義する:

$$\rho^{\mathrm{inc}}_{ik}(J_i^{\sigma_{\mathrm{lopt}}}) = p_{ik}(J_i^{\sigma_{\mathrm{lopt}}})/b_{ik},$$

$$\rho^{\mathrm{dec}}_{ik}(J_i^{\sigma_{\mathrm{lopt}}}) = \begin{cases} -1, & p_{ik}(J_i^{\sigma_{\mathrm{lopt}}}) = 0, \\ 0, & p_{ik}(J_i^{\sigma_{\mathrm{lopt}}}) \neq 0, \end{cases}$$

$$\phi^{\mathrm{inc}}_{i}(J_i^{\sigma_{\mathrm{lopt}}}) = q_{i}(J_i^{\sigma_{\mathrm{lopt}}}),$$

$$\phi^{\mathrm{dec}}_{i}(J_i^{\sigma_{\mathrm{lopt}}}) = \begin{cases} -1, & q_{i}(J_i^{\sigma_{\mathrm{lopt}}}) = 0, \\ 0, & q_{i}(J_i^{\sigma_{\mathrm{lopt}}}) \neq 0. \end{cases}$$

ペナルティ重みの更新は, 前回ペナルティ重みを更新した後, 局所探索中に移動した解の中に実行可能解があったか否か, および資源制約と個数制約のいずれかを満たす解が存在したか否かによって決定するものとした. 以下にそれぞれの場合に対する更新方法を詳しく述べていく.

ケース 1: 直前の局所探索中に移動した解の中に資源制約のみを満たす解も個数制約のみを満たす解も存在しなかった場合.

任意の $i \in I, k \in K$ に対して

$$\Gamma_{ik} = \begin{cases} size\_inc\_\alpha \cdot \dfrac{\rho_{ik}^{\mathrm{inc}}(J_i^{\sigma_{\mathrm{lopt}}})}{\max\limits_{i' \in I, k' \in K} |\rho_{i'k'}^{\mathrm{inc}}(J_{i'}^{\sigma_{\mathrm{lopt}}})|}, \\ \qquad \max\limits_{i' \in I, k' \in K} |\rho_{i'k'}^{\mathrm{inc}}(J_{i'}^{\sigma_{\mathrm{lopt}}})| > 0, \\ 0, \qquad \text{その他}, \end{cases}$$

$$\Delta_i = \begin{cases} size\_inc\_\beta \cdot \dfrac{\phi_i^{\mathrm{inc}}(J_i^{\sigma_{\mathrm{lopt}}})}{\max\limits_{i' \in I} |\phi_{i'}^{\mathrm{inc}}(J_{i'}^{\sigma_{\mathrm{lopt}}})|}, \\ \qquad \max\limits_{i' \in I} |\phi_{i'}^{\mathrm{inc}}(J_{i'}^{\sigma_{\mathrm{lopt}}})| > 0, \\ 0, \qquad \text{その他}, \end{cases}$$

を用いてペナルティ重み α_{ik} と β_i を

$$\alpha_{ik} := \alpha_{ik}(1 + \Gamma_{ik}), \tag{9}$$
$$\beta_i := \beta_i(1 + \Delta_i) \tag{10}$$

と増加させる．ここで，$size\_inc\_\alpha < 0$ と $size\_inc\_\beta > 0$ はパラメータである．

ケース 2: 直前の局所探索中に移動した解の中にいずれか一方の制約のみを満たす解が存在した場合．
以下のように各ペナルティ重み α_{ik} と β_i の一方を増加させる．資源制約のみを満たす解が存在し，個数制約を満たす解が存在しなかったときは

$$\alpha_{ik} := \alpha_{ik}, \tag{11}$$
$$\beta_i := \beta_i(1 + \Delta_i) \tag{12}$$

とし，一方，個数制約のみを満たす解が存在し，資源制約を満たす解が存在しなかったときは

$$\alpha_{ik} := \alpha_{ik}(1 + \Gamma_{ik}), \tag{13}$$
$$\beta_i := \beta_i \tag{14}$$

とする．ここで，Γ_{ik} と Δ_i はケース 1 で定義したものである．

ケース 3: 直前の局所探索中に移動した解の中に資源制約のみを満たす解と個数制約のみを満たす解がそれぞれ存在した場合．
すべての $i \in I$ と $k \in K$ に対してペナルティ重み α_{ik} と β_i を

$$\alpha_{ik} := \alpha_{ik}(1 + size\_inc\_factor \cdot \Gamma_{ik}),$$
$$\beta_i := \beta_i(1 + size\_inc\_factor \cdot \Delta_i)$$

と増加させる．ここで，Γ_{ik} と Δ_i はケース 1 で定義したものであり，また，$0 < size\_inc\_factor \leq 1$ はパラメータである．

ケース 4: 直前の探索中に移動した解の中に実行可能解が存在した場合．
各ペナルティ重み α_{ik}，β_i を減少させる．更新はケース 1 と同様の方法で行う．ただし，$size\_inc\_\alpha$ と $size\_inc\_\beta$ の代わりに $size\_dec\_\alpha$ と $size\_dec\_\beta$ ($0 < size\_dec\_\alpha < 1$, $0 < size\_dec\_\beta < 1$ はパラメータ) を用い，$\rho_{ik}^{\mathrm{inc}}(J_i^{\sigma_{\mathrm{lopt}}})$ と $\phi_i^{\mathrm{inc}}(J_i^{\sigma_{\mathrm{lopt}}})$ の代わりに $\rho_{ik}^{\mathrm{dec}}(J_i^{\sigma_{\mathrm{lopt}}})$ と $\phi_i^{\mathrm{dec}}(J_i^{\sigma_{\mathrm{lopt}}})$ を用いる．

3.5 ランダムな摂動

本節では，ランダムな摂動について述べる．反復局所探索法では，解が局所最適解 σ_{lopt} に辿り着いたとき，解にランダムな摂動を加え，解を変更する．そしてその解を初期解とし，再び局所探索を行う．このランダムな摂動は局所探索とは異なり，改悪解への移動も許可し通常の局所探索では到達できない解への移動を目的としている．

提案アルゴリズムでは，ランダムな摂動としてシフトキックと交換キックの 2 つを用いた．これらの動作は解が局所最適解 σ_{lopt} にたどり着き，ペナルティ重み α_{ik} と β_i の更新を行った直後に行う．

まず，シフトキックについて述べる．提案手法では解の評価は制約違反に応じたペナルティをコストに付加したものを評価値としている．そのため，個数制約のペナルティの値を縦軸に取り，あるエージェント i に割り当てられた仕事数を横軸に取ってグラフを描くと，各 g に対して h_{ig} と $h_{i,g+1}$ の間に山が出来る．このペナルティの山を越える解への移動は局所探索では起こりにくい．そこで，あるエージェントに割り当てられた仕事数に山を越えるような大きな変更を加えるため，シフトキックを提案した．シフトキックとは，ランダムに選ばれた任意のエージェント $i_1 \in I$ に割り当てられている複数の仕事の割当先を変更する操作である．ここで，エージェント i_1 に現在割り当てられている仕事数に最も近い割り当て可能な仕事数を h_{i_1,g^*} とし，シフトキックを行った後，エージェント i_1 に割り当てられている仕事数を h'_{i_1} とする．これらを以下のように定義する:

$$h'_{i_1} = \begin{cases} h_{i_1,g^*-1}, & g^* \geq 2, \\ h_{i_1,g^*}, & g^* = 1, \end{cases}$$
$$g^* = \arg \min_{g'=1,2,\ldots,|H_{i_1}|} \{||J_{i_1}^{\sigma}| - h_{i_1 g'}|\}.$$

エージェント i_1 から $|J_{i_1}^{\sigma}| - h'_{i_1}$ 個の仕事をランダムに選び，割当先を変更する．このようにしてエージェント i_1 に割り当てられている仕事数を強制的に h'_{i_1}

個にすることが出来る．なお，複数の仕事の割当先を同一のエージェント $i_2 \in I$ に変更すると，資源制約を違反しやすくなり，割当に偏りが生じるため，割当先を変更する仕事ごとに割当先をランダムに選ぶ．シフトキック後の解に対してシフト近傍の探索を行うと，シフトキックによって割当先が変更された仕事が元のエージェントに戻ってしまう可能性がある．そのため，シフトキックによって仕事が移動した割当先のエージェント集合 $I' \subset I$ を保持し，I' のいずれかのエージェント i' に含まれている仕事に関する交換近傍による探索を近傍内に改善解がなくなるまで行う．この操作は各エージェントに割り当てられた仕事の数を変更することなくペナルティ付きコスト $pcost$ の改良を行うため，その後のシフト近傍の探索によって仕事の割当先を元に戻してしまうことを防ぐ効果が期待出来る．

次に交換キックについて述べる．これは各エージェントに割り当てられた仕事の数を変えることなく解の移動を行うために用いた．交換キックとは，異なるエージェントに割り当てられたふたつの仕事 j_1, j_2 $(\in J, \sigma(j_1) \neq \sigma(j_2))$ をランダムに選び，それらの割当先を交換する操作を複数回行うことである．ここで，一度交換を行った仕事はそれ以降のキックの対象から除き，交換キックの操作によって割当先を変更された仕事の中に重複はないものとする．交換キックによる摂動の大きさは交換を行う回数に影響される．そのため交換を行う回数をパラメータ γ とし，本研究では γ の値をキックのたびに 2 から 5 の整数から選ぶこととした．これをシフト交換キック (SSK) と呼ぶ．

3.6 提案アルゴリズム全体の枠組み

本節では，提案アルゴリズムの枠組みをまとめる．まず，初期解 σ_{start} をランダムに作成し，局所探索を行う．局所探索ではシフト近傍，交換近傍，連鎖シフト近傍を用いて解の改善を行っていく．ここで，交換近傍内の探索は，現在の解のシフト近傍内に改善解がない場合に限って行う．同様に，連鎖シフト近傍はシフト近傍と交換近傍内のいずれにも改善解がない場合に限って探索を行う．探索が局所最適解 σ_{lopt} にたどり着くたびに，ペナルティ重み α_{ik} と β_i の更新を行う．ここで，過去の局所最適解の中で良い解を σ_{seed} (初期値は $\sigma_{\mathrm{seed}} = \sigma_{\mathrm{start}}$) とし，更新後の α_{ik} と β_i を用いたペナルティ付きコストが $pcost(\sigma_{\mathrm{lopt}}) \leq pcost(\sigma_{\mathrm{seed}})$ を満たす場合は，$\sigma_{\mathrm{seed}} := \sigma_{\mathrm{lopt}}$ と更新する．σ_{seed} に対してシフトキックまたは交換キックをランダムに選び行う．シフトキックを行った場合は，シフト近傍操作で元の局所最適解に戻ってしまう事があるため，交換近傍の探索から始めることでそのようなサイクリングを防ぐ．これにより得られた新たな初期解に対して同様に局所探索を行う．終了条件は計算時間があらかじめ定めた上限を超えたときとし，解の移動が行われるたびに終了条件を満たしているか否かをチェックする．

提案アルゴリズム (SSK-CS)

Step1 解 σ をランダムに生成する．$\sigma_{\mathrm{seed}} := \sigma$ とする．

Step2 終了条件を満たしている場合，暫定解を出力して終了する．

Step3 即時移動戦略によるシフト近傍内の探索を行う．もし，$pcost(\sigma') < pcost(\sigma)$ となる解 $\sigma' \in N_{\mathrm{shift}}(\sigma)$ を発見したら，$\sigma := \sigma'$ とし，Step2 へ．

Step4 即時移動戦略による交換近傍内の探索を行う．もし，$pcost(\sigma') < pcost(\sigma)$ となる解 $\sigma' \in N_{\mathrm{swap}}(\sigma)$ を発見したら，$\sigma := \sigma'$ とし，Step2 へ．

Step5 アルゴリズム SCS を用いた連鎖シフト近傍の探索を行う．もし，$pcost(\sigma') < pcost(\sigma)$ となる解 $\sigma' \in N_{\mathrm{chain}}(\sigma)$ を発見したら，$\sigma := \sigma'$ とし，Step2 へ．

Step6 3.4 節に述べた方法でペナルティ重み α_{ik} と β_i の更新を行う．

Step7 $pcost(\sigma) \leq pcost(\sigma_{\mathrm{seed}})$ ならば，$\sigma_{\mathrm{seed}} := \sigma$ とする．

Step8 σ_{seed} に対してアルゴリズム SSK を用いてランダムな摂動を加えることによって得られた解を σ とし，Step2 へ．

4 計算実験

ここでは，提案アルゴリズム (SSK-CS) と次のアルゴリズムとの比較を行う: (1) 汎用ソルバー CPLEX, (2) 提案アルゴリズムから連鎖シフト近傍を除いたアルゴリズム (SSK-noCS). SSK-CS と SSK-noCS は C 言語にて実装した．SSK-noCS は連鎖シフト近傍

表1　タイプCの問題例に対する計算結果

n	m	s	SSK-CS		SSK-noCS		CPLEX	
			Best	TTB(秒)	Best	TTB(秒)	Best	TTB(秒)
100	5	1	1942[b]	31.81	1942[b]	10.08	1942[a]	4.15
100	5	8	1954[b]	11.77	1954[b]	15.37	1954[a]	17.57
100	10	1	1421[b]	58.34	1421[b]	41.13	1421[a]	122.36
100	10	8	1473[b]	48.55	1476	59.90	1475	274.94
100	20	1	1321	59.01	1329	19.04	1317[b]	91.27
100	20	8	1350[b]	153.13	1373	41.05	1351	244.47
200	5	1	3462[b]	14.80	3462[b]	368.40	3462[a]	29.51
200	5	8	3483[b]	208.85	3483[b]	549.23	3486	360.50
200	10	1	2829	32.05	2827[b]	433.84	2830	589.90
200	10	8	2853[b]	173.94	2855	50.11	-	-
200	20	1	2433	39.25	2432[b]	406.88	-	-
200	20	8	2442[b]	179.05	2461	145.44	-	-
400	10	1	5609	930.63	5612	1021.85	5604[b]	640.52
400	10	8	5655	658.96	5648	1178.08	5636[b]	1177.86
400	20	1	4814[b]	1148.71	4837	1166.66	-	-
400	20	8	4860[b]	974.29	4866	863.36	-	-
400	40	1	4288[b]	911.05	4321	1084.30	-	-
400	40	8	4312[b]	1161.71	4374	1166.98	-	-

表2　タイプDの問題例に対する計算結果

n	m	s	SSK-CS		SSK-noCS		CPLEX	
			Best	TTB(秒)	Best	TTB(秒)	Best	TTB(秒)
100	5	1	6371	297.68	6365	88.10	6360[b]	18.72
100	5	8	6444	197.09	6458	195.41	6435[b]	129.10
100	10	1	6414	68.91	6402	127.63	6395[b]	254.91
100	10	8	6554	115.80	6555	211.17	6541[b]	229.05
100	20	1	6308[b]	171.49	6354	179.63	6345	287.33
100	20	8	6518[b]	225.89	6576	209.10	6604	160.69
200	5	1	12769	122.54	12769	299.22	12753[b]	439.35
200	5	8	12833	218.39	12828	279.19	12787[b]	217.96
200	10	1	12507	512.22	12534	94.58	12472[b]	447.61
200	10	8	12602	492.17	12664	185.29	12570[b]	557.40
200	20	1	12431[b]	497.25	12448	532.40	-	-
200	20	8	12610[b]	133.18	12682	340.70	-	-
400	10	1	25080	326.09	25107	232.04	24994[b]	1094.17
400	10	8	25234	846.17	25283	1185.88	25083[b]	1161.00
400	20	1	24775[b]	1115.42	24842	858.41	-	-
400	20	8	25057[b]	1014.05	25172	1199.80	-	-
400	40	1	24668[b]	1181.31	24872	1038.97	-	-
400	40	8	24984[b]	899.74	25264	1156.47	-	-

の探索を除いたこと以外は提案アルゴリズムと同様である．また計算環境は Dell Precision 470, Xeon 3GHz, 2MB cache, 1GB memory である．問題例は，次の2つの方法により作成した: (1) 多資源一般化割当問題のベンチマーク問題例に個数制約をランダムに加える，(2) 一般化割当問題のベンチマーク問題例の仕事数 400 の問題例に資源制約と個数制約をランダムに加える．

まず，多資源一般化割当問題のベンチマーク問題例を利用した生成方法について述べる．多資源一般化割当問題の問題例のタイプはタイプCからEまであり，これら全てのタイプを用いた．個数制約を以下のように生成した．ランダムに選んだひとつのエージェントを $i' \in I$ とし，それ以外の各エージェント $i \in I \setminus \{i'\}$ に対して以下を行う．まず，h_{i1} を $[1,5]$ からランダムに選び，$g := 2$ とする．次に，整数 y を $[1,5]$ からランダムに選んで，$h_{ig} := h_{i,g-1} + y$ としたのち $g := g+1$ とする操作を $h_{ig} \geq 2n/m$ となるまで繰り返す．はじめにランダムに選んだひとつのエージェント $i' \in I$ に対しては，次のように個数制約を設けた．i' に割当可能な仕事数の下限として l を $[n/(4m), n/(2m)]$ からランダムに選び，上限

表 3　タイプ E の問題例に対する計算結果

n	m	s	SSK-CS		SSK-noCS		CPLEX	
			Best	TTB(秒)	Best	TTB(秒)	Best	TTB(秒)
100	5	1	12710^b	88.15	12710^b	229.30	12710^a	7.26
100	5	8	12823^b	77.43	12823^b	130.15	12845	215.78
100	10	1	11711	250.88	11697^b	153.21	11780	185.52
100	10	8	11904^b	167.79	11909	72.03	11951	230.21
100	20	1	8734^b	186.85	8779	175.77	8801	285.45
100	20	8	11935^b	87.02	12060	164.46	12281	294.03
200	5	1	24956	171.68	24958	500.96	24951^a	23.51
200	5	8	25040	122.03	25033^b	308.46	25046	580.04
200	10	1	23431	520.06	23426^b	36.40	23468	307.69
200	10	8	23615	402.84	23592	444.45	23437^b	562.26
200	20	1	22725^b	583.79	22784	580.24	-	-
200	20	8	23160^b	540.78	23352	37.10	24616	588.54
400	10	1	45800^b	1185.16	45805	1088.76	45876	1194.00
400	10	8	45955	630.84	45998	977.27	45877^b	392.46
400	20	1	45154^b	1020.18	45187	785.82	-	-
400	20	8	45319^b	1043.52	45465	742.50	46745	1188.60
400	40	1	46088	1012.45	45653^b	1119.02	-	-
400	40	8	45588^b	796.96	46066	1066.49	-	-

として r を $[n/(2m), 2n/m]$ からランダムに選んで $H_{i'} = \{l, l+1, \ldots, r\}$ とする．i' を用意する理由は，全てのエージェントに対して i' 以外のエージェントと同じ生成方法を採ってしまうと，個数制約を満たす解が存在しない問題例となってしまう可能性があり，これを回避するためである．

次に，一般化割当問題のベンチマーク問題例を利用した生成方法について述べる．一般化割当問題の問題例のタイプはタイプ A から E まで 5 つの種類があるが，それらのうちタイプ A と B は比較的簡単な問題傾向であるため，タイプ C から E までの 3 つを用いた．まず，資源制約の生成方法を以下に述べる．各 $i \in I$, $j \in J$, $k \in K \setminus \{1\}$ に対して $a_{ijk} = 3a_{ij1}/4 + \delta_{ijk}a_{ij1}/2$ とした．ここで δ_{ijk} は $[0,1]$ からランダムに選ぶ．また，$b_{ik} = 0.8\sum_{j\in J} a_{ijk}/m$ とした．個数制約は多資源一般化割当問題のベンチマーク問題例に対して行った生成方法と同様の方法で生成した．

以上からタイプ C, D, E, $n \in \{100, 200\}$, $m \in \{5, 10, 20\}$, $s \in \{1, 8\}$ からなる 36 個の問題例と，タイプ C, D, E, $n \in \{400\}$, $m \in \{10, 20, 40\}$, $s \in \{1, 8\}$ からなる 18 個の問題例の，合計 54 個の問題例を作成し，計算実験を行った．

ペナルティ重みの初期値は $\alpha_{\text{init}} = 1$, $\beta_{\text{init}} = 1$ とし，重みの更新時の各パラメータは $size\_inc\_\alpha = 0.1$, $size\_dec\_\alpha = 0.1$, $size\_inc\_\beta = 0.5$, $size\_dec\_\beta = 0.1$, $size\_inc\_factor = 0.1$ とした．また，終了条件は計算時間があらかじめ定めた上限 t を超えたときとし，計算時間の条件 t は仕事の数 n に比例するように $t = 3n$ とした．

表 1–3 は SSK-CS, SSK-noCS, CPLEX の計算結果である．表中の「n」は仕事数，「m」はエージェント数，「s」は資源の種類数を表している．また，表中の「Best」は計算時間内に各アルゴリズムによって得られた最も良い実行可能解の目的関数値を示し，「TTB」は最も良い実行可能解が得られたときの計算時間 (秒) を示している．表中の出力解に「b」と記されているのは比較した 3 つのアルゴリズムの中で最も良い解であることを示しており，CPLEX の出力解に「a」と記されているのはその解が問題の厳密な最適解であることを示している．表中の「ー」は計算時間内に実行可能解が得られなかったことを示している．

これらの表から次のことが観測される:

- SSK-CS はほとんどの問題において SSK-noCS よりも良い結果を得ることが出来た．このことから連鎖シフト近傍の有効性を確認することが出来た．
- SSK-CS はタイプ C と E の問題例の多くに対して，CPLEX よりも良い解を得ることが出来た．また，タイプ D の問題例に対しては，CPLEX は 10 問に対して最良解を得ているのに対し，提案手法の SSK-CS は 8 問で最良解を得ている．このように最良解を得た個数に関しては CPLEX の方がやや勝っているも

のの, 仕事数が大きい問題例や, 仕事数に対するエージェント数の割合が大きい問題例に対しては提案手法の方が勝っている傾向にある. このことから提案アルゴリズムはとくに規模の大きな問題例に対してより効果的であると考えられる.

5 まとめ

個数制約付き多資源一般化割当問題の計算の複雑さを明らかにしたのち, 発見的解法を提案した. まず, 計算の複雑さについては, とくにこの問題が個数制約のみを有する場合に対する計算の複雑さについて考え, 個数制約が特別な条件を満たす場合は多項式時間で解けるが, 一般の場合はNP困難であることを示した. 次に, この問題に対して反復局所探索法に基づくアルゴリズムを提案した. ベンチマーク問題例を元に作成した問題例に対して計算実験を行ったところ, 多くの問題例に対して汎用ソルバー CPLEX より良い結果を得ることが出来た. とくに, 大規模な問題例においては, 調べた全てのタイプの問題例に対して汎用ソルバー CPLEX より平均的に良い結果を得ることが出来た.

参考文献

[1] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice-Hall, 1993.

[2] W. Cook, A. Rohe, Computing minimum-weight perfect matchings. INFORMS J. Comput. 11 (1999) 138–148.

[3] J. Edmonds, Maximum matching and a polyhedron with (0,1) vertices, J. Res. Nat. Bur Standards 69B (1965) 125–130.

[4] H.N. Gabow, Data structures for weighted matching and nearest common ancestors with linking, Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (1990) 434–443.

[5] M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.

[6] B. Gavish, H. Pirkul, Algorithms for the multi-resource generalized assignment problem, Management Science 37 (1991) 695–713.

[7] E.L. Lewler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, 1976.

[8] S. Lin, B.W. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Oper. Res. 21 (1973) 498–516.

[9] K. Mehlhorn, G. Schäfer, Implementation of $O(nm \log n)$ weighted matchings in general graphs: the power of data structures. In: Algorithm Engineering; WAE-2000; LNCS 1982 (S. Näher, D. Wangner, eds.), pp.23–38; also electronically in the ACM Journal of Experimental Algorithmics 7 (2002) 138–148.

[10] R.M. Nauss, Solving the generalized assignment problem: an optimizing and heuristic approach, INFORMS J. Comput. 15 (2003) 249–266.

[11] S. Sahni, T. Gonzalez, P-complete approximation problems, J. ACM 23 (1976) 555–565.

[12] P.M. Thompson, J.B. Orlin, The theory of cyclic transfers, Working Paper No. OR 200-89, Operations Res. Center, MIT, Cambridge, 1989.

[13] M. Yagiura, T. Ibaraki, F. Glover, An ejection chain approach for the generalized assignment problem, INFORMS J. Comput. 16 (2004) 133–151.

[14] M. Yagiura, T. Ibaraki, F. Glover, A path relinking approach with ejection chains for the generalized assignment problem, European J. Oper. Res. 169 (2006) 548–569.

[15] M. Yagiura, S. Iwasaki, T. Ibaraki, F. Glover, A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem, Discrete Optimization 1 (2004) 87–98.